PCL Services

Thor Design Panel 2/3

November 26, 1997

Version 2.2

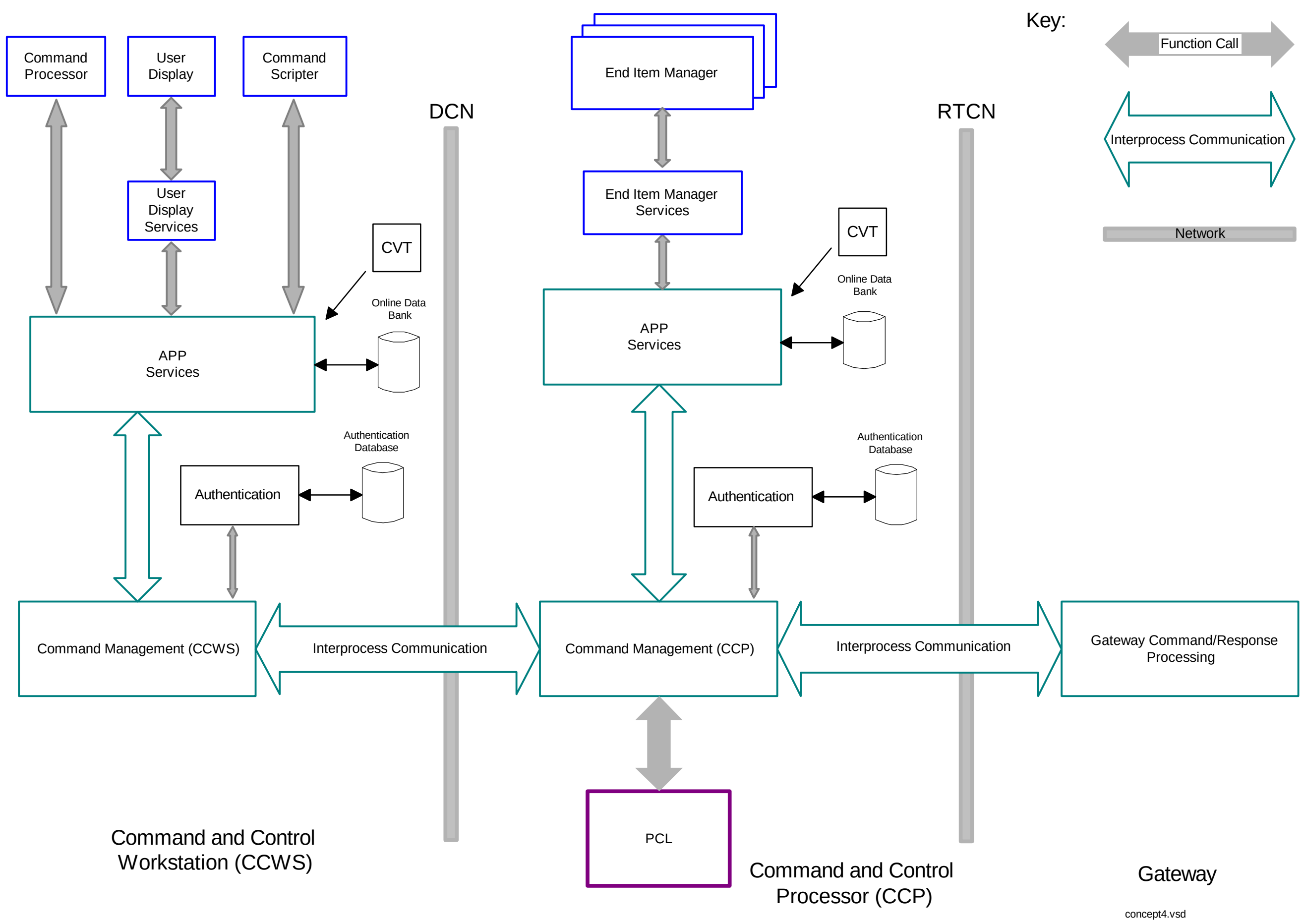
PCL Services Design Panel 2/3
Table of Contents

1. Prerequisite Control Logic

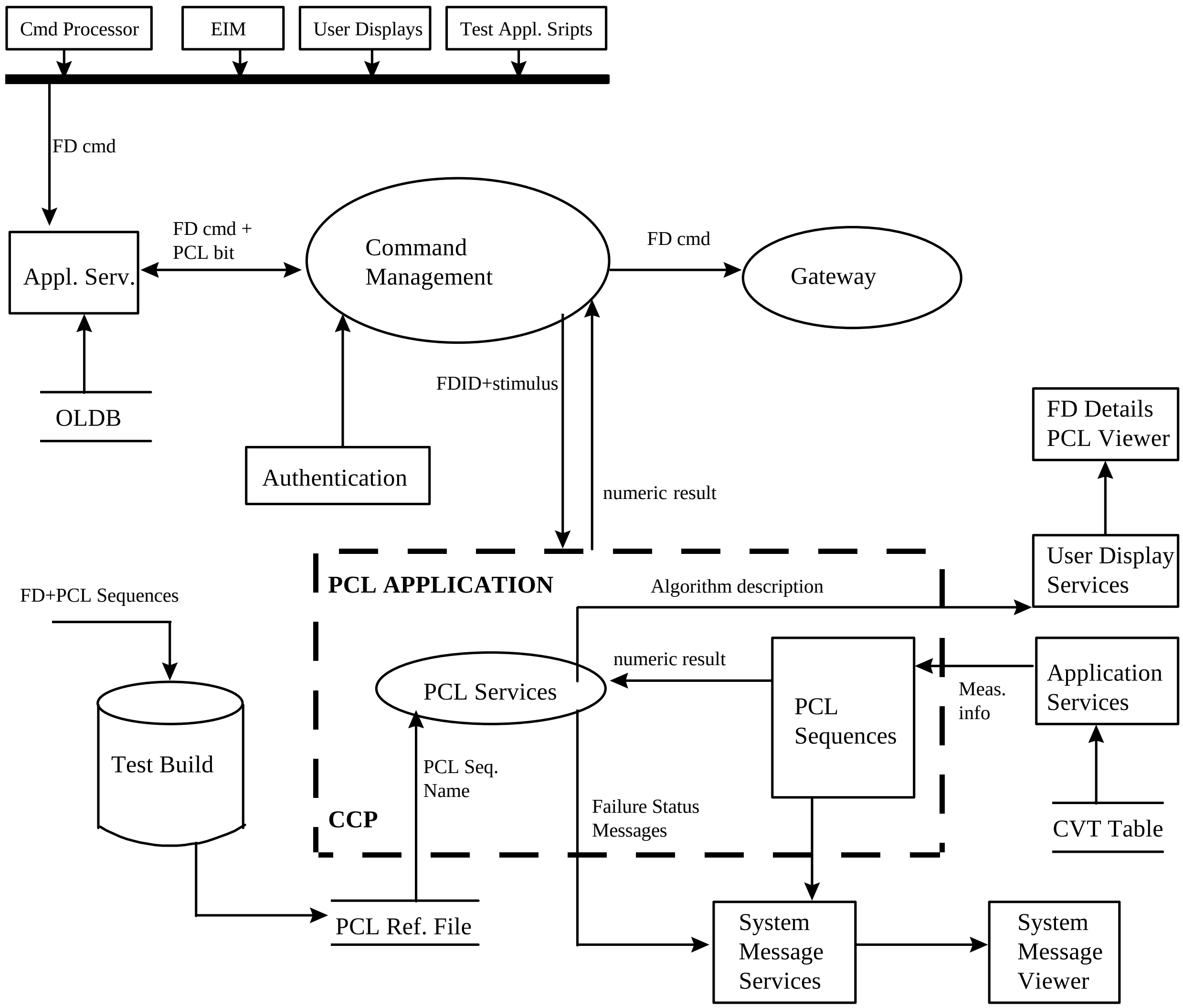
1.1 Prerequisite Control Logic Introduction

1.1.1 Prerequisite Control Logic Overview

Prerequisite Control Logic (PCL) applications exist to prevent users from issuing commands when end items are not properly configured. PCL protects against commands issued from any and all application types, including the Command Processor, User Displays, End Item Managers, and Test Application Scripts. PCL also provides a level of command safety that supports contingency or off-nominal situations requiring manual command inputs at the FD level. PCL shall exist as a separate function that independent of End Item Management. PCL provides *additional* safety margin above and beyond that provided by other application software (i.e., End Item Management).



1.1.2 Prerequisite Control Logic Operational Description



1 Set Up

The PCL sequences are user pre-defined C++ objects that test for the criteria for safe operation. The PCL sequences and their initiating FDs are available to the test build process for inclusion in the TCID and the PCL Reference File. The PCL reference file should contain only FDIDs and the names of the associated PCL sequences. The Test Build process should fail if the PCL sequences cannot be found, do not have an associated initiating FD, or fail to compile/link.

Each FD command that requires PCL execution would have its PCL bit set and the Override bit not set. The PCL bit for every FD command can be looked up in the On Line Data Bank. The Override bit can be set from the Command Processor or the Reactive Control Logic. The PCL and the Override bits are included in the C-to-C packet header.

2 Initialization

Prior to Test Build, as each PCL sequence is written, an entry is made to the PCL association database which contains the FD name, and the name of the PCL sequence object name with which it is associated.

PCL Application is a single process that contains the PCL Services and all PCL sequences. The PCL sequences are linked into the PCL Application process as library objects. During Test Build, PCL Services is compiled and linked with this library. A flat file with the data from the PCL association database, the PCL reference file, is also passed to Ops CM for initialization.

During initialization, the constructor function for each sequence passes the name of each sequence and its object pointer to PCL Services. PCL Services uses these passed parameters and the PCL reference file to create its own association table which matches each FDID to its PCL sequence object pointer.

3 Operation

FD command is issued from either the Command Processor, User Displays, End item managers, or Test Application Scripts to Command Management through Application Services. Command Management determines from the information in the C-to-C packet header whether to send the command to PCL Application or out to the Gateway.

When PCL Application receives an FD command, it executes the PCL sequence and returns a numeric value to Command Management. This value, if zero, indicates that it is safe to issue the initiating FD stimulus value. Non zero returned values may be used as “reason codes” that communicate to the initiating application the reason for PCL failure. The “reason codes” are predefined in the PCL Application and are passed to Command Management. The PCL application execution is performed on the CCP.

Limited Application Services calls are permitted inside a PCL sequence such as: sending out applicable PCL failure messages, obtaining the initiating stimulus FD and its value, current measurement FD values, pseudo FD values, and health information.

1.2 Prerequisite Control Logic Specifications

1.2.1 Prerequisite Control Logic Groundrules

- PCL applications are implemented as C++ objects.
- Test Build is responsible to build and integrate PCL Reference file into a TCID. Reporting PCL failure during build is also the responsibility of Test Build.
- Application Software IPTs are responsible for defining and developing PCL sequences.
- PCL and Override bits are included in the header of the C-to-C packet.
- PCL Application is not able to issue any commands directly.
- Modification of a PCL sequence requires only a partial TCID rebuild and reload. However, the addition or deletion of a PCL sequence affects Test Build process significantly.
- Calls to PCL Applications should return within an appropriate time (i.e. there should not be any infinite loops).
- Command Management sends PCL failure messages to System Message Services.
- PCL Application sends service failure messages to System Message Services.
- PCL Application should run atomically.
- *DBSAFE and Test Build will ensure that PCL does not get initiated for a pseudo FD.*

1.2.2 Prerequisite Control Logic Functional Requirements

Prerequisite Control Logic SLS Requirements

(SLS 2.2.3.3.2) The CLCS shall provide the capability to protect from inadvertent issuance of commands.

1. Prerequisite logic
2. Two step protocol required on critical commands entered from the keyboard (e.g., arm, execute, and disarm logic)

(SLS 2.2.5.5) Prerequisite Control Logic

To protect against inadvertently issuing a command which could injure personnel or damage equipment, CLCS provides the capability for the user to predefine logic that verifies the appropriate conditions are met prior to executing the command.

(SLS 2.2.5.5.1) RTPS shall provide the capability to verify that certain conditions are met before issuing any FD command to an End-Item.

(SLS 2.2.5.5.2) The RTPS Prerequisite Control Logic function shall be fault tolerant.

(SLS 2.2.5.5.3) RTPS shall provide the capability to manually override Prerequisite Control Logic once command issuance has been blocked.

(SLS 2.2.5.5.4) RTPS shall provide the capability for Reactive Sequence Test applications to bypass Prerequisite Control Logic.

Prerequisite Control Logic Derived Requirements

1. PCL Application only executes the sequence identified for the initiating stimulus FD from the PCL Reference file.

2. PCL Reference file is allowed to associate a sequence to more than one FD.

3. PCL Application returns a numeric value to Command Management such as,
   - A zero indicates that it is safe to issue the initiating FD stimulus value.
   - A non zero indicates a predefined reason code for the failure.

4. The association between a PCL sequence and its initiating FD(s) are in a data product accessible to Test Build.

5. The Command Processor can override PCL at runtime.

6. PCL sequences provide the pointers to the PCL sequence algorithm description string to System Viewer through Application Services.

7. PCL application execution is performed on the CCP.

8. *PCL is integrated into the CLCS System Integrity scheme. Any system or application processes required for PCL operation is monitored for health, checkpointed, and restarted if necessary.*

9. *RTPS provides the capability to generate a runtime report that details all PCL related information for the TCID. The contents of the report includes, but not be limited to: FD name and nomenclature, PCL required flag, PCL application name, and dynamic PCL status (active/inhibited).*

1 File Repository Management

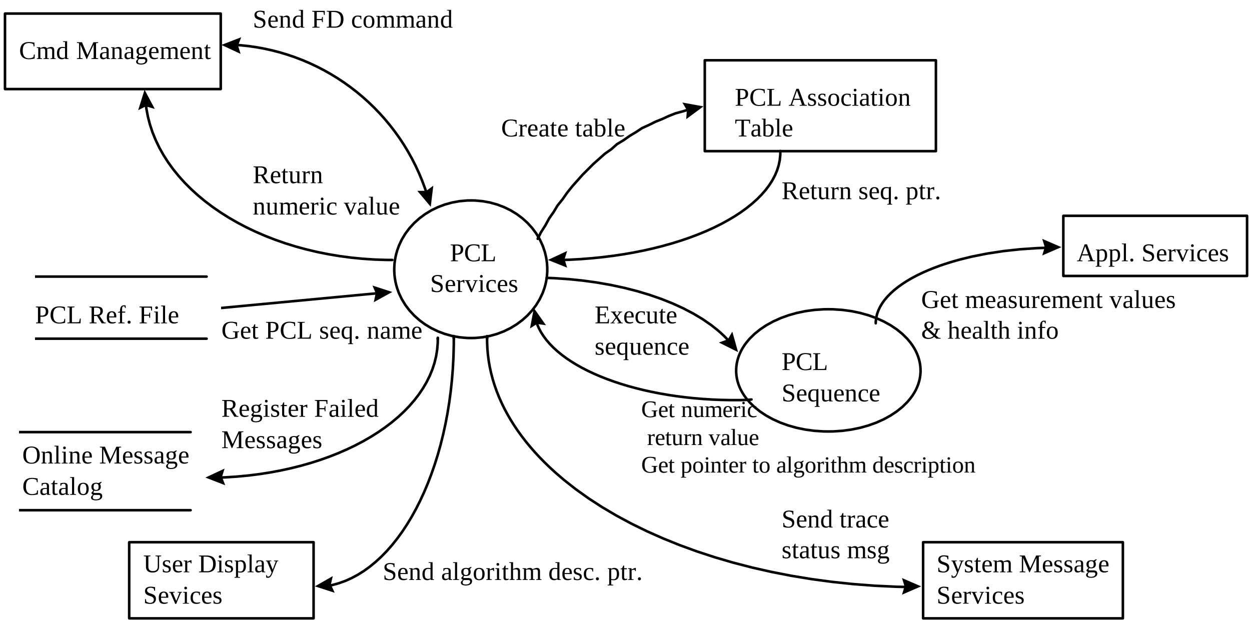
1.1 The user defined logic for each PCL application will be maintained in a separate header and implementation file.

1.2 The user source code for each PCL application will reside in the CM Repository.

1.2.3 Prerequisite Control Logic Performance Requirements

Prerequisite Control Logic performance requirements are not defined. The PCL Service shall process a PCL request in a period of time that does not impact the requirements specified in section 3.2.3 of the Command Support CSCI Requirements and Design Specification , 84K00550.

1.2.4 Prerequisite Control Logic Interfaces Data Flow Diagrams



- PCL Reference File provides the FDIDs and the names of their associated PCL sequences.
- Command Management passes the FD command and expects a numeric return value.
- Application Services provides interfaces for PCL Applications to get current measurement values and health information.
- Online Message Catalog registers PCL failed status message.
- System Message Services and User Display Services provide interfaces for PCL Application to send failed status messages and the pointer to the PCL algorithm description.
- PCL Services provides the following functions
  - create PCL Association table.
  - get FDID and stimulus from Command Management.
  - get PCL sequence pointer from PCL Association table.
  - execute PCL sequence and capture numeric return value.
  - return numeric value to Command Management.
  - send failed status message and pointer to algorithm description to System message Services.
- PCL Sequence is a set of user pre-defined logic that would return a numeric value to PCL Services. During execution, PCL sequence can obtain current measurements, values, and health information.

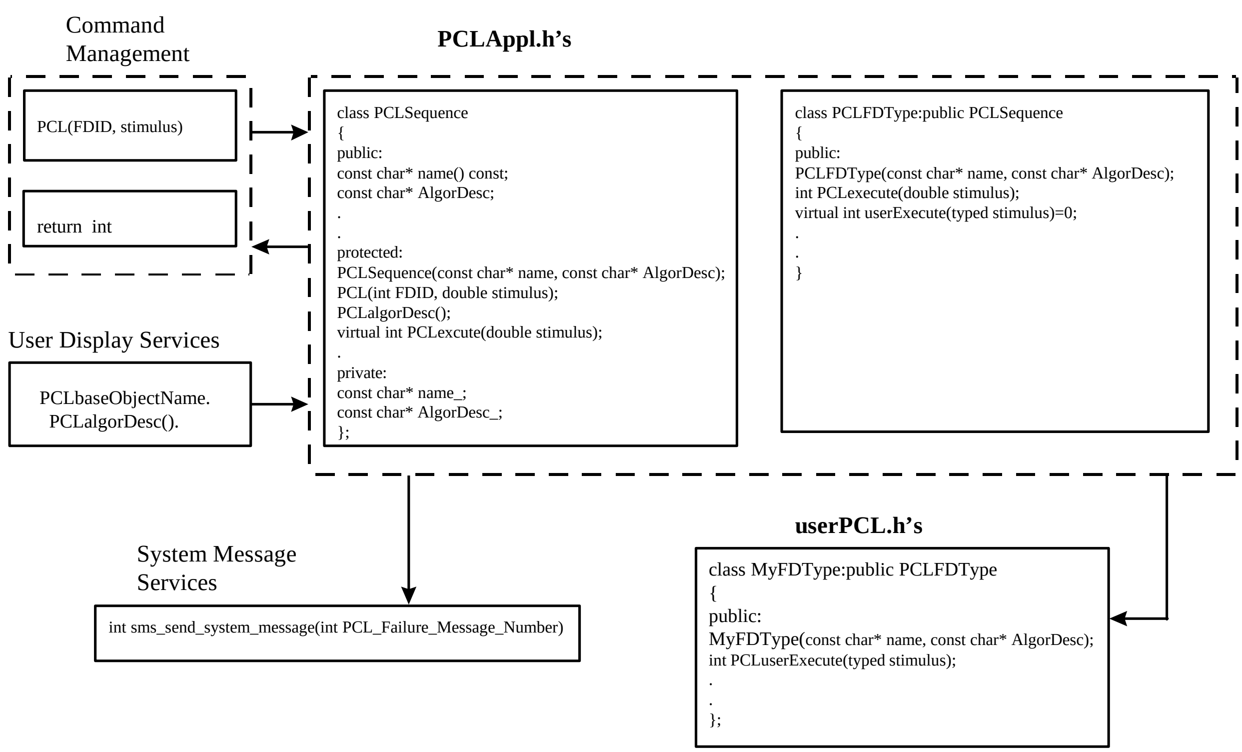
1.3 Prerequisite Control Logic Design Specifications

PCL Application will perform and provide these following functions:

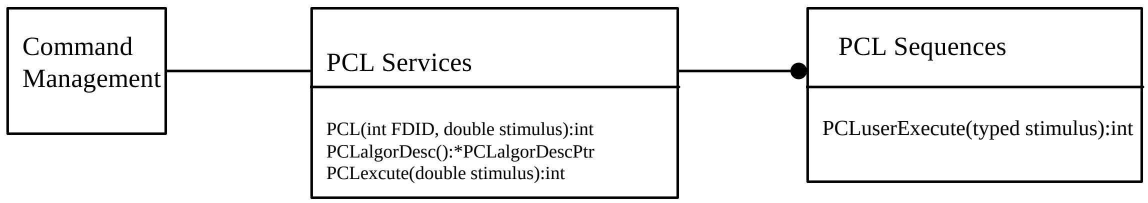
1. Create PCL association tables.
2. Receive FDID and stimulus for PCL sequence execution.
3. Execute PCL sequence and capture a numeric return value.
4. Pass the numeric return value to Command management.
5. Output reason codes when command is rejected

1.3.1 Prerequisite Control Logic Detailed Data Flow

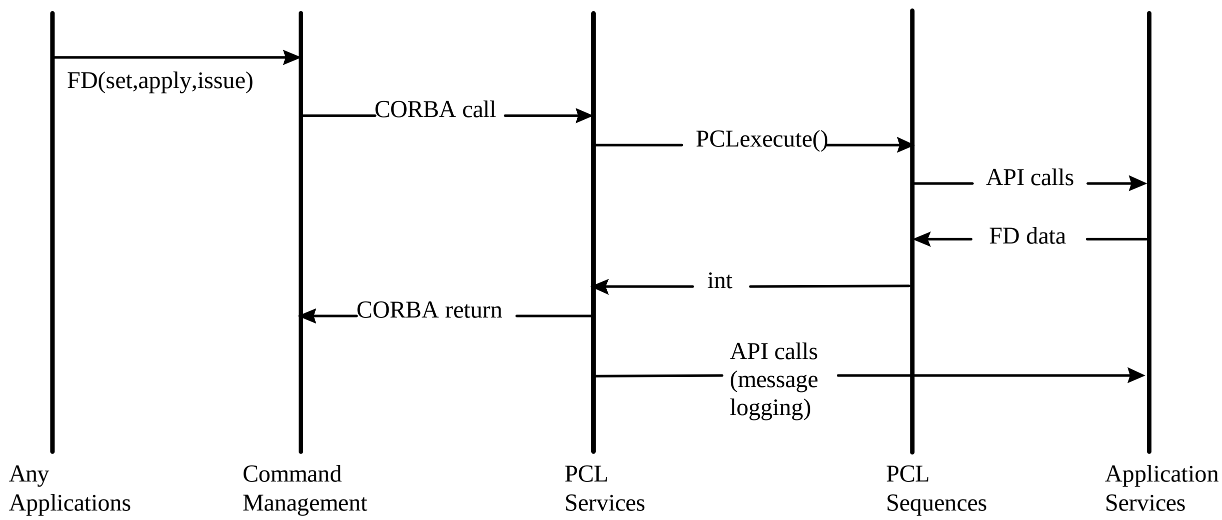
This data flow provides a pictorial representation of the data flow between external sources and destinations and the major and minor functions of the PCL Application.



Prerequisite Control Logic Object Diagram



Prerequisite Control Logic Scenario Diagram



1.3.2 Prerequisite Control Logic External Interfaces

1.3.2.1 Prerequisite Control Logic Message Formats

The interface between Command Management and PCL Applications is a standard IDL-formatted message compliant with OMG CORBA 2.0.

1.3.2.2 Prerequisite Control Logic Display Formats

PCL display will be done by System Message Viewer CSC.

1.3.2.3 Prerequisite Control Logic Input Formats

- PCL Reference File is in ASCII format and contains FDIDs and PCL Sequence Names in two separate columns.
- Input from Command Management contains the FDID and a two to eight bytes long stimulus.

1.3.2.4 Recorded Data

Name of Recorded Data	Recording Type	SDC	Local
PCL Association Failure	System Message	X	
PCL Logic Execution Failure	System Message	X	

1.3.2.5 Prerequisite Control Logic Printer Format

N/A

1.3.2.6 Interprocess Communications (C-to-C Communications)

CORBA calls will be utilized between Command Management and PCL. If CORBA requires excessive processing overhead, then IPC calls will be implemented.

1.3.2.7 Prerequisite Control Logic External Interface Calls

- CORBA call from Command Management: PCLFDTypeObjectName.PCL(FDID, stimulus).
- System Message Services API call from PCL Application: int sms_send_system_message(int PCL_Failure_Message_Number).
- User Display Services API call to PCL Application: PCLbaseObjectName.PCLalgorDesc(FDID).

1.3.2.8 Prerequisite Control Logic Table Formats

PCL Reference File format:

FDID1	PCLSeqName1<CR>
FDID2	PCLSeqName2<CR>

PCL Association table:

FDIDs	PCL Sequence Names	PCL Sequence Pointers
FDID	PCLSeqName	PCLSeqPtr
“	“	“
“	“	“
“	“	“

1.3.2.9 Prerequisite Control Logic Startup Script

PCL Application requires both the PCL Reference File and PCL Sequences during initialization.

Command Management must be started and registration made to PCL interface’s CORBA ORB prior to starting the PCL Application. Command Management must block any PCL-directed stimulus FDs until the PCL Application is loaded and connected.

To start the PCL Application process, execute PCL with the single argument of the PCL Reference File. The PCL Application object constructor will register it with the interface’s CORBA ORB, and instantiate the PCL sequence objects.

1.3.3 Prerequisite Control Logic Test Plan

The specific test cases that will be run include:

1. Load and create the PCL association table.
2. For each FD type:
   - Send an FD command that resulted in a true return after PCL sequence execution.
   - Send an FD command that resulted in a false return after PCL sequence execution and verify that the proper failure message is logged.
3. Send an FD command that resulted in PCL sequence is not found.
4. Verify through System Viewer that it displays the correct algorithm description.

PREREQUISITE CONTROL LOGIC ISSUES

- PCL failure messages are not defined at this point. These messages will be defined and registered with Online Message Catalog.
- Strong typed checking at build time is yet to be finalized.

Appendix A

Below are examples of a FD type dm_gse_on_off header and C files for user codes:

gcl17.h

```
#ifndef __GCL17_H__
#define __GCL17_H__

#include "pcl_dm_gse_on_off.h"
#include "fd_types.h"

extern DM_GSE_ON_OFF GLOX1073E;
extern DM_GSE_ON_OFF GLOX0072E;

extern PD_ON_OFF NLOK3221X;
extern PD_ON_OFF NLOK3341X;
extern PD_ON_OFF NLOX1073E;
extern PD_ON_OFF NLOX0072E;

class GCL17 : public VPCL_DM_GSE_ON_OFF
{
public:
  GCL17(const char* name, const char* algorithmDescription);
  ~GCL17();
  int userExecute(ON_OFF initiatingStimulusValue);
};

#endif
```

gcl.C

```
#include "gcl17.h"

static const char* name = "GCL17";
static const char* algorithmDescription =
"THIS PROGRAM PREVENTS CLOSING THE TRANSFER LINE FILL\n\
 VALVE A86461 UNLESS THE EXPRESSION\n\
 ((A OR ((B OR C) AND D)) AND (((E OR F) AND (G OR H))\n\
 OR (I AND J AND K) OR L OR M) OR N) IS SATISFIED WHERE:\n\
\n\
   A.  PUMPS A126 AND A127 OFF\n\
   B.  A134 OPEN\n\
   C.  A196 OPEN\n\
   D.  A102 OPEN\n\
   E.  A75120 CLOSED\n\
   F.  A75121 CLOSED\n\
   G.  A75127 CLOSED\n\
   H.  A75128 CLOSED\n\
   I.  PV9 OPEN\n\
   J.  PV10 OPEN\n\
   K.  PD1 OPEN\n\
   L.  A86483 OPEN\n\
   M.  A86483 OPEN\n\
   N.  A86460 OPEN\n\
\n\
IT ALSO PREVENTS OPENING THE VALVE UNLESS THE EXPRESSION \n\
((A OR D OR (E AND F AND G)) AND H) OR(I AND J) IS SATISFIED \n\
WHERE:\n\
   A.  PUMPS A126 AND A127 OFF\n\
   D.  A86462 OPEN\n\
   E.  PV9 OPEN\n\
   F.  PV10 OPEN\n\
   G.  PD1 OPEN\n\
   H.  ET 100% LEVEL LESS THAN 10% WET\n\
   I.  A134 OPEN\n\
   J.  A102 OPEN";

GCL17::GCL17(const char* name, const char* algorithmDescription) :
  VPCL_DM_GSE_ON_OFF(name, algorithmDescription)
{
}

GCL17::~GCL17()
{
}

///////////////////////////////////////////////////////////////////////////
//
//  NOTE:  THE FOLLOWING EXAMPLE IS A STRAIGHT PORT OF PCL SEQUENCE
//  GCL17. IT DOES NOT USE THE FACILITIES OF DATA HEALTH OR DATA FUSION
//  TO SIMPLIFY THE LOGIC FLOW.  IT IS ASSUMED THAT IN PRACTICE DATA HEALTH
//  AND DATA FUSION WOULD BE USED TO REDUCE THE AMOUNT OF LOGIC THAT IS
//  EXPRESSED IN TERMS OF FUNCTION DESIGNATORS.
//
```

```
/////////////////////////////////////////////////////////////////////////

int
GCL17::userExecute(ON_OFF initiatingStimulus)
{
  /////////////////////////////////////////////////////////////////////////
  //
  //  BEGIN USER CODE
  //
  /////////////////////////////////////////////////////////////////////////

  result_t result = FAILURE;

  bool a = false;
  bool b = false;
  bool c = false;
  bool d = false;
  bool e = false;
  bool f = false;
  bool g = false;
  bool h = false;
  bool i = false;
  bool j = false;
  bool k = false;
  bool l = false;
  bool m = false;
  bool n = false;

  if (initiatingStimulus == ON)
    {
      //  Perform valve close checks

      //  A.  A126 and A127 OFF

      if ((NLOK3221X.value() == OFF) && (NLOK3341X.value() == OFF))
             a = true;
      else
             a = false;

      //  B.  A134 OPEN

      if (((NLOX1073E.value() == ON) || (GLOX1073E.value() == ON)) &&
               ((NLOX0072E.value() == ON) || ((NLOX0072E.value() == OFF) &&
                                                    (GLOX0072E.value() == OFF))))
             b = true;
      else
             b = false;

      //  (other checks omitted for brevity)

      if ((a || ((b || c) && d)) && (((e || f) && (g || h)) ||
                                              (i && j && k) || l || m) || n)
             result = SUCCESS;
      else
             result = FAILURE;
```

```
    }
  else
    {
      //  Perform valve open checks

      //  (other checks omitted for brevity)

      if (((a || d | (e && f && g)) && h) || (i && j))
              result = SUCCESS;
      else
              result = FAILURE;
    }

  return result;

  ////////////////////////////////////////////////////////////////////////////
  //
  //  END USER CODE
  //
  ////////////////////////////////////////////////////////////////////////////
}

GCL17 gcl17(name, algorithmDescription);
```